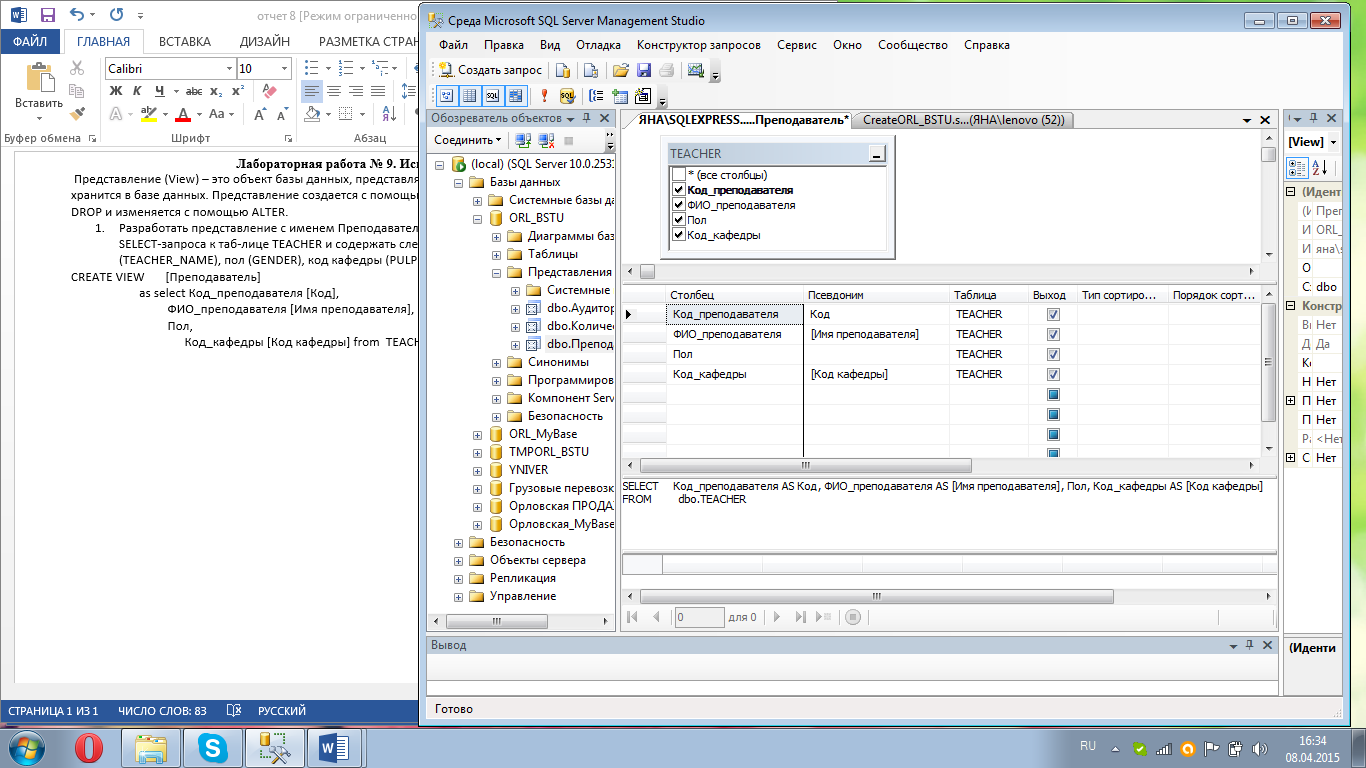
**Лабораторная работа № 9. Использование представлений**

Представление (View) – это объект базы данных, представляющий собой поименованный SELECT-запрос, который хранится в базе данных. Представление создается с помощью оператора CREATE, удаляется с помощью оператора DROP и изменяется с помощью ALTER.

1. Разработать представление с именем Преподаватель. Представление должно быть построено на основе SELECT-запроса к таблице TEACHER и содержать следующие столбцы: код (TEACHER), имя преподавателя (TEACHER\_NAME), пол (GENDER), код кафедры (PULPIT).

CREATE VIEW [Преподаватель]

as select Код\_преподавателя [Код],

ФИО\_преподавателя [Имя преподавателя], Пол,

Код\_кафедры [Код кафедры] from TEACHER;

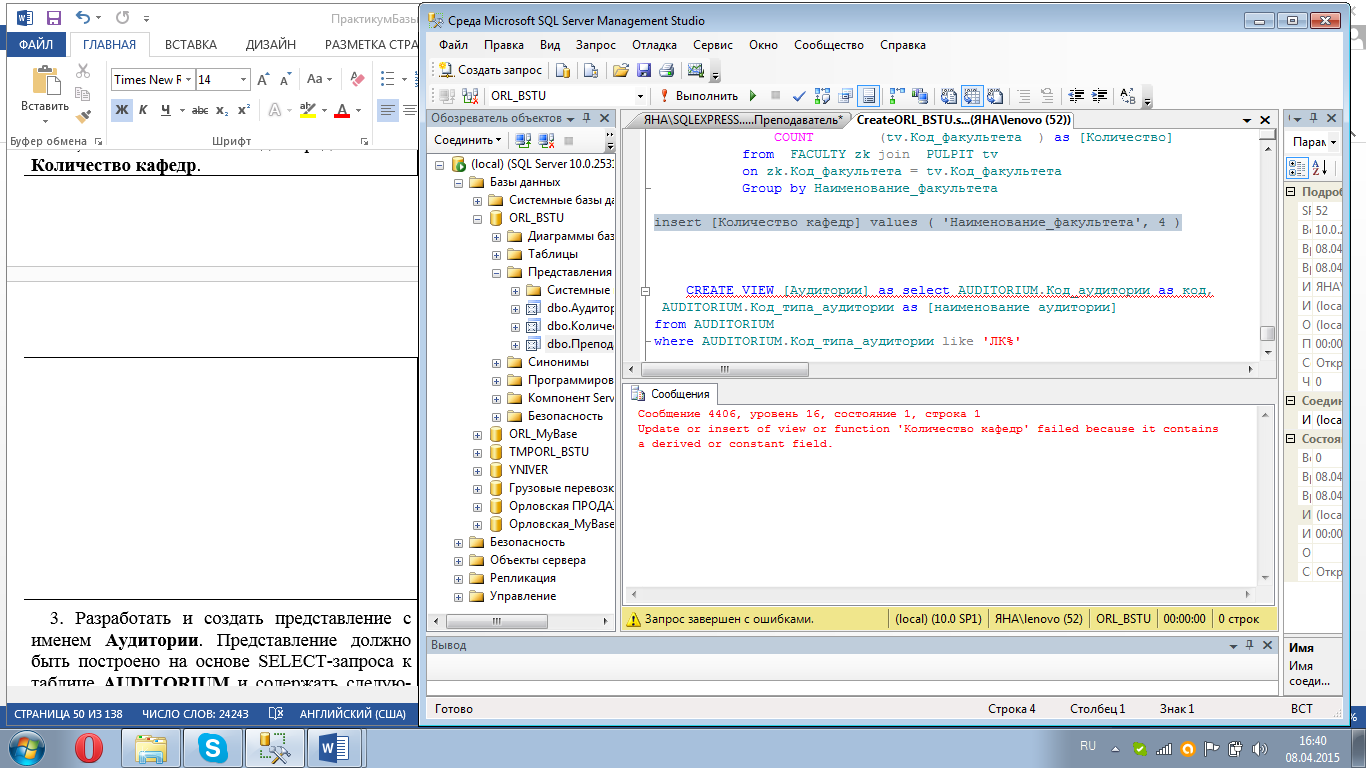
1. Разработать и создать представление с именем Количество кафедр, к таблицам FACULTY и PULPIT. Представление должно содержать столбцы: факультет (FACULTY.FACULTY\_ NAME), количество кафедр (вычисляется на основе строк таблицы PULPIT). Объяснить невозможность выполнения операторов INSERT, UPDATE и DELETE для представления Количество кафедр.

CREATE VIEW [Количество кафедр] as SELECT zk.Наименование\_факультета [Факультет],

COUNT (tv.Код\_факультета) as [Количество]

from FACULTY zk join PULPIT tv on zk.Код\_факультета = tv.Код\_факультета

Group by Наименование\_факультета;

** insert [Количество кафедр] values ( 'Наименование\_факультета', 4 )**

При создании представлений, позволяющих выполнять операции INSERT, DELETE и UPDATE, базовый SELECT-запрос должен удовлетворять правилам:

 запрос не должен содержать секцию группировки GROUP BY;

 запрос не должен применять агрегатные функции, опции DISTINCT и TOP, операторы UNION, INTERSECT и EXCEPT;

 в SELECT-списке запроса не должно быть вычисляемых значений;

 в секции FROM запроса должна указываться только одна таблица.

В приведенном выше примере представление содержит группировку GROUP BY и в секции FROM две таблицы, поэтому и происходит ошибка.

1. Разработать и создать представление с именем Аудитории, к таблице AUDITORIUM и содержать следующие столбцы: код (AUDITORIUM), наименование аудитории (AUDITORIUM\_NAME). Представление должно отоб-ражать только лекционные аудитории (в столбце AUDITORIUM\_ TYPE строка, начинающаяся с символа ЛК). Представление должно допускать выполнение оператора INSERT, UPDATE и DELETE.

CREATE VIEW [Аудитории] as

select AUDITORIUM.Код\_аудитории as код, AUDITORIUM.Код\_типа\_аудитории as [наименование аудитории]

from AUDITORIUM where AUDITORIUM.Код\_типа\_аудитории like 'ЛК%';

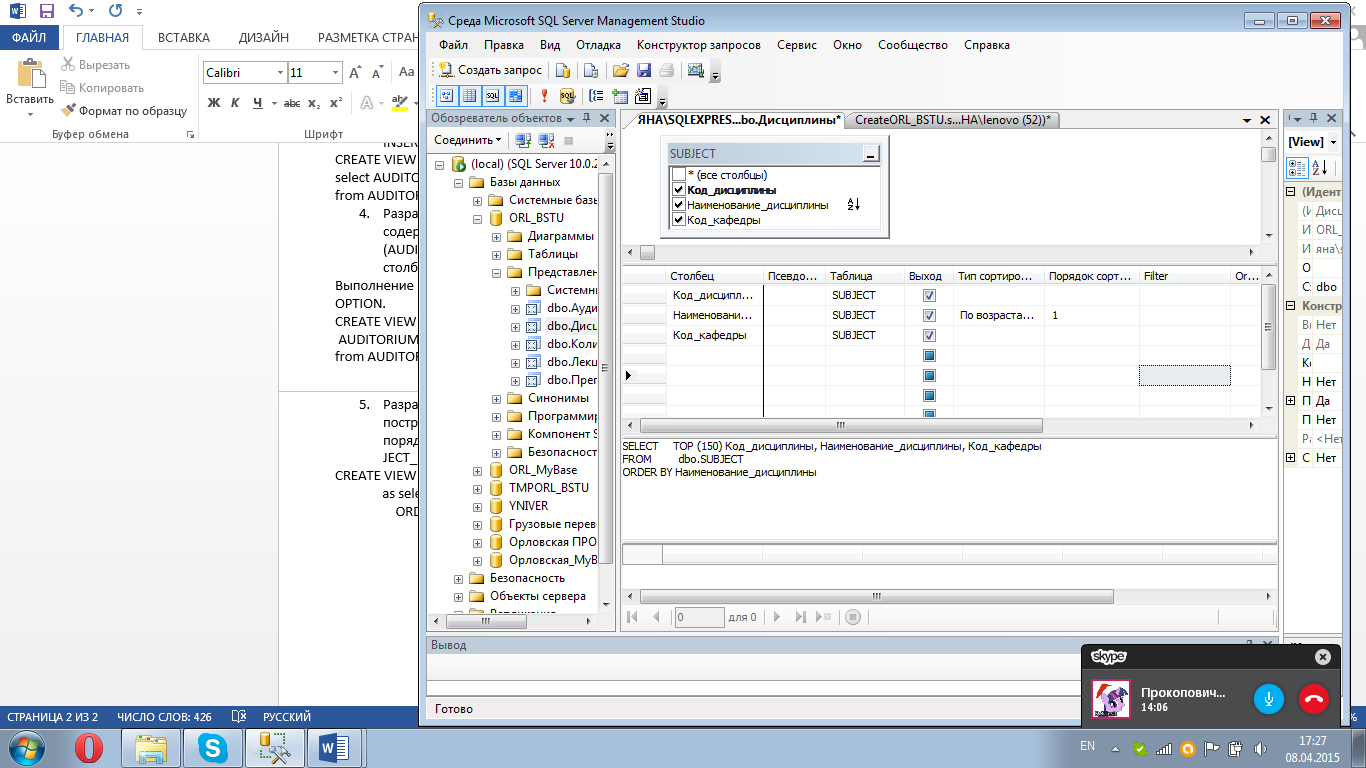
1. Разработать и создать представление с именем Лекционные\_аудитории, к таблице AUDITORIUM и содержать следующие столбцы: код (AUDITORIUM), наименование аудитории (AUDITORIUM\_NAME). Представление должно отображать только лекционные аудитории (в столбце AUDITORIUM\_TYPE строка, начинающаяся с символов ЛК).

Выполнение INSERT и UPDATE допускается, но с учетом ограничения, задаваемого опцией WITH CHECK OPTION.

CREATE VIEW [Лекционные аудитории] as select AUDITORIUM.Код\_аудитории as код,

AUDITORIUM.Код\_типа\_аудитории as [наименование аудитории]

from AUDITORIUM where AUDITORIUM.Код\_типа\_аудитории like 'ЛК%' WITH CHECK OPTION;

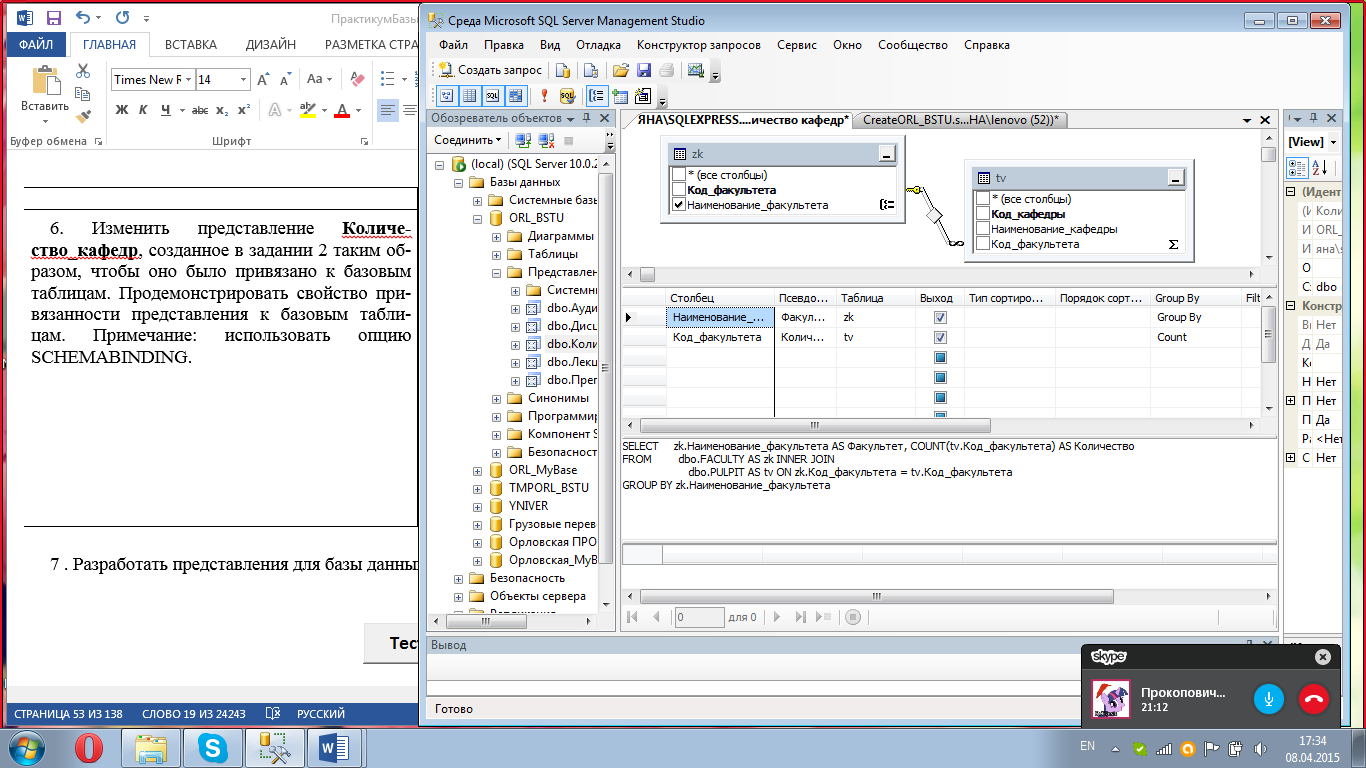
1. Разработать и создать представление с именем Дисциплины, к таблице SUBJECT, отображать все дисци-плины в алфавитном порядке и содержать следующие столбцы: код (SUBJECT), наименование дисциплины (SUB-JECT\_NAME) и код кафедры (PULPIT). Примечание: использовать секции TOP и ORDER BY.

CREATE VIEW Дисциплины

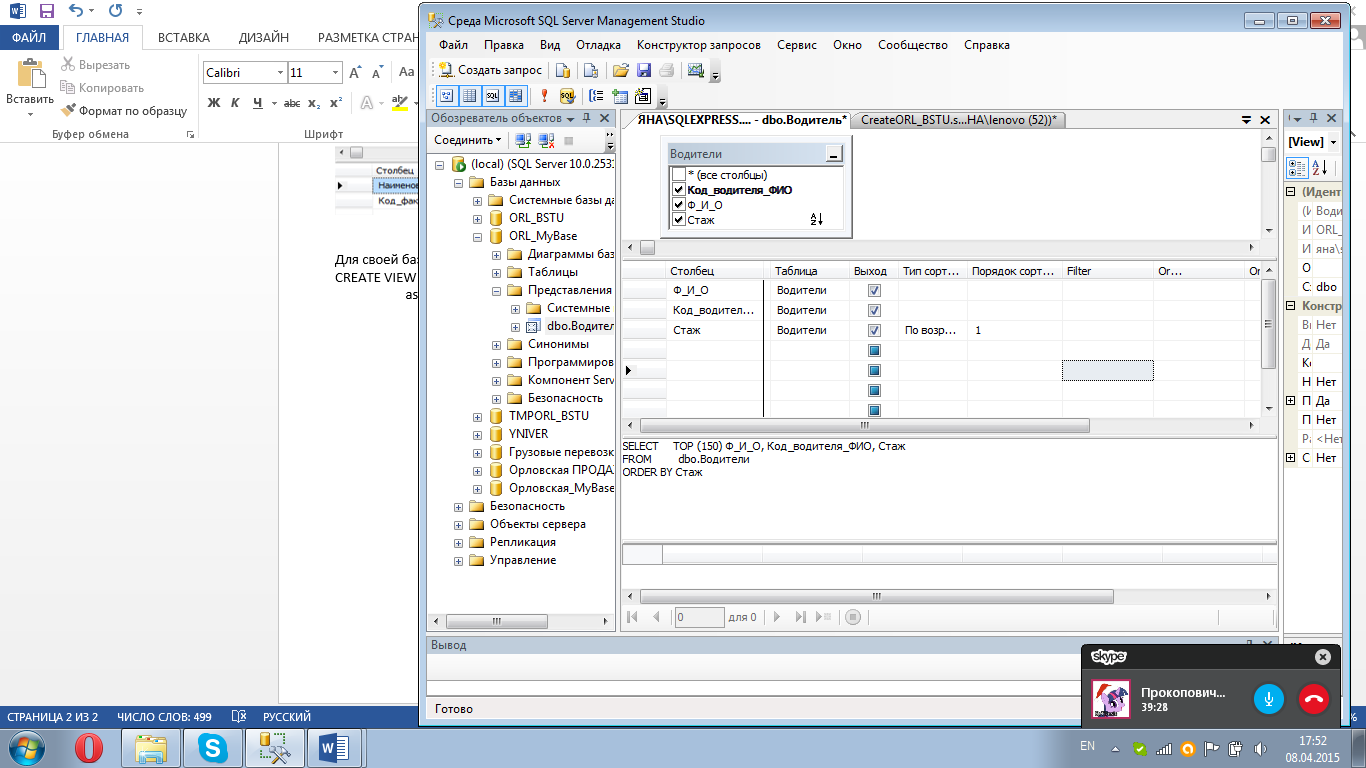
as select TOP 150 Код\_дисциплины, Наименование\_дисциплины, Код\_кафедры from SUBJECT

ORDER BY Наименование\_дисциплины;

1. Изменить представление Количество\_кафедр, созданное в задании 2 таким образом, чтобы оно было привязано к базовым таблицам. Продемонстрировать свойство привязанности представления к базовым таблицам. Примечание: использовать опцию SCHEMABINDING.

Опция SCHEMABINDING устанавливает запрещение на операции с таблицами и представлениями, которые могут привести к нарушению ра-ботоспособности представления.

При использовании опции SCHEMABINDING требуется использовать в SELECT-запросе для имен таблиц и представлений двухкомпонентный формат (в имени присутствует точка).

**Для своей базы:**

CREATE VIEW [Водитель]

as select Код\_водителя\_ФИО [Код],

Ф\_И\_О [Имя водителя],

Стаж from Водители;

Немного модифицируем ( по возрастанию):

Alter VIEW Водитель

as select TOP 150 Ф\_И\_О, Код\_водителя\_ФИО, Стаж from Водители

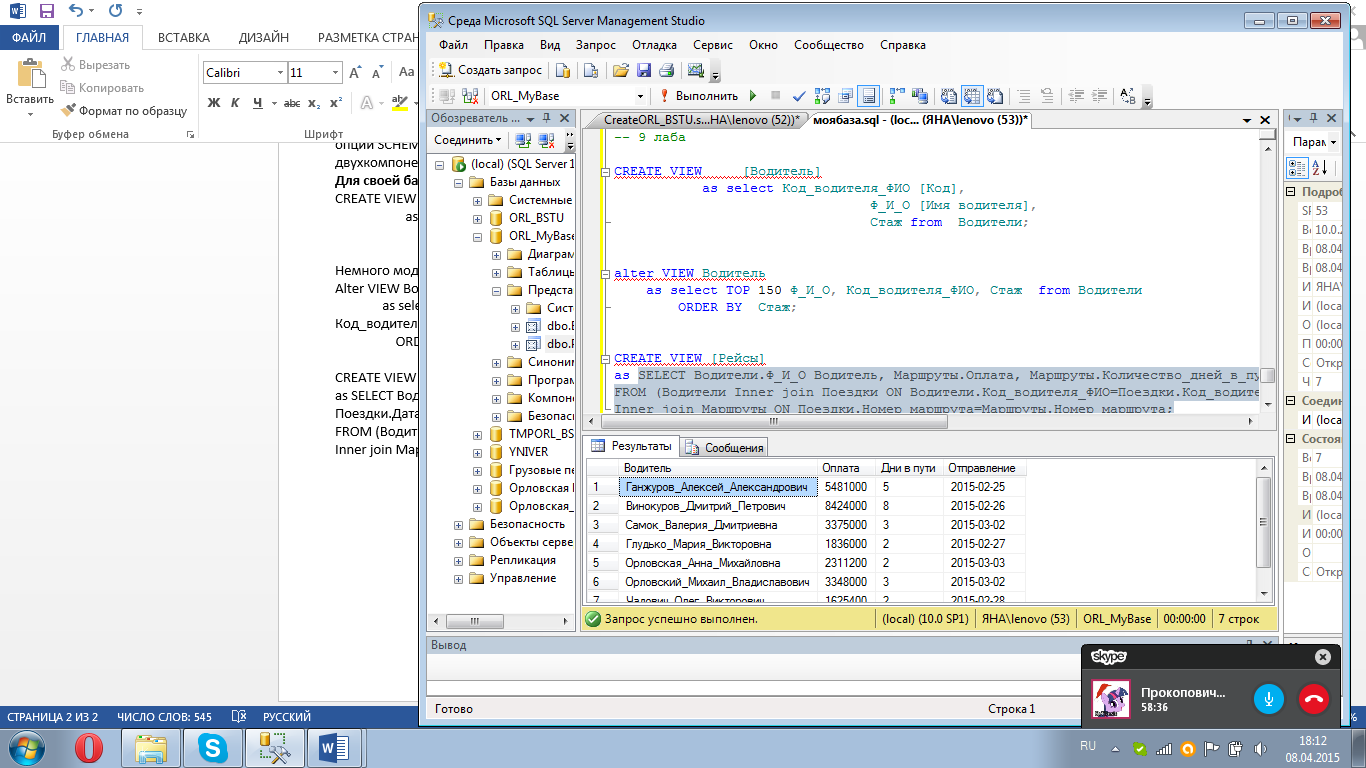
ORDER BY Стаж;

CREATE VIEW [Рейсы]

as SELECT Водители.Ф\_И\_О Водитель, Маршруты.Оплата, Маршруты.Количество\_дней\_в\_пути [Дни в пути], Поездки.Дата\_отправления [Отправление]

FROM (Водители Inner join Поездки ON Водители.Код\_водителя\_ФИО=Поездки.Код\_водителя\_ФИО)

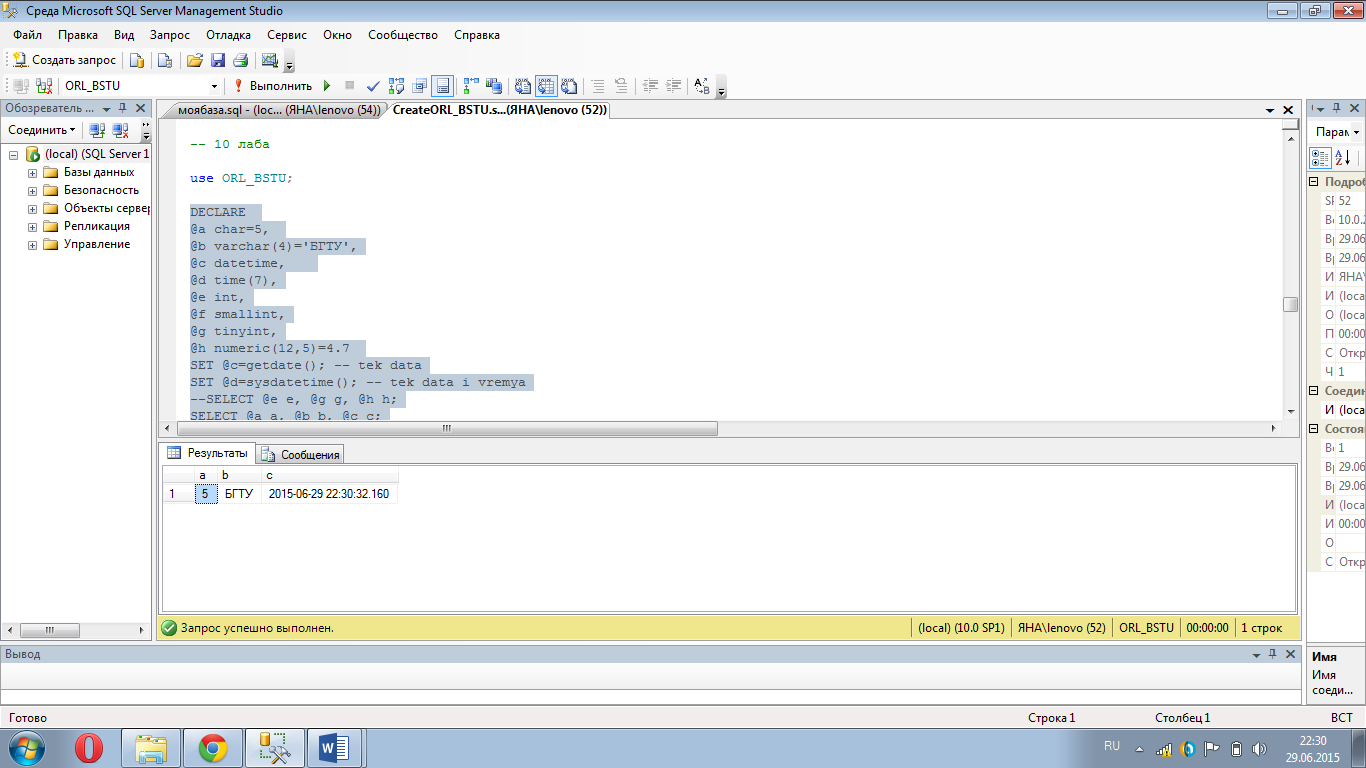
Inner join Маршруты ON Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута;



**Лабораторная работа № 10. Основы программирования на T-SQL**

Для объявления переменных, используемых в программах, предназначен оператор DECLARE. Имя переменной должно начинаться с символа @. С помощью оператора SET можно переменной присвоить значение и выполнять вычисления. Оператор SELECT позволяет нескольким переменным присвоить значения. Функция CAST используется для преобразования типов.

DECLARE

@a char=5,

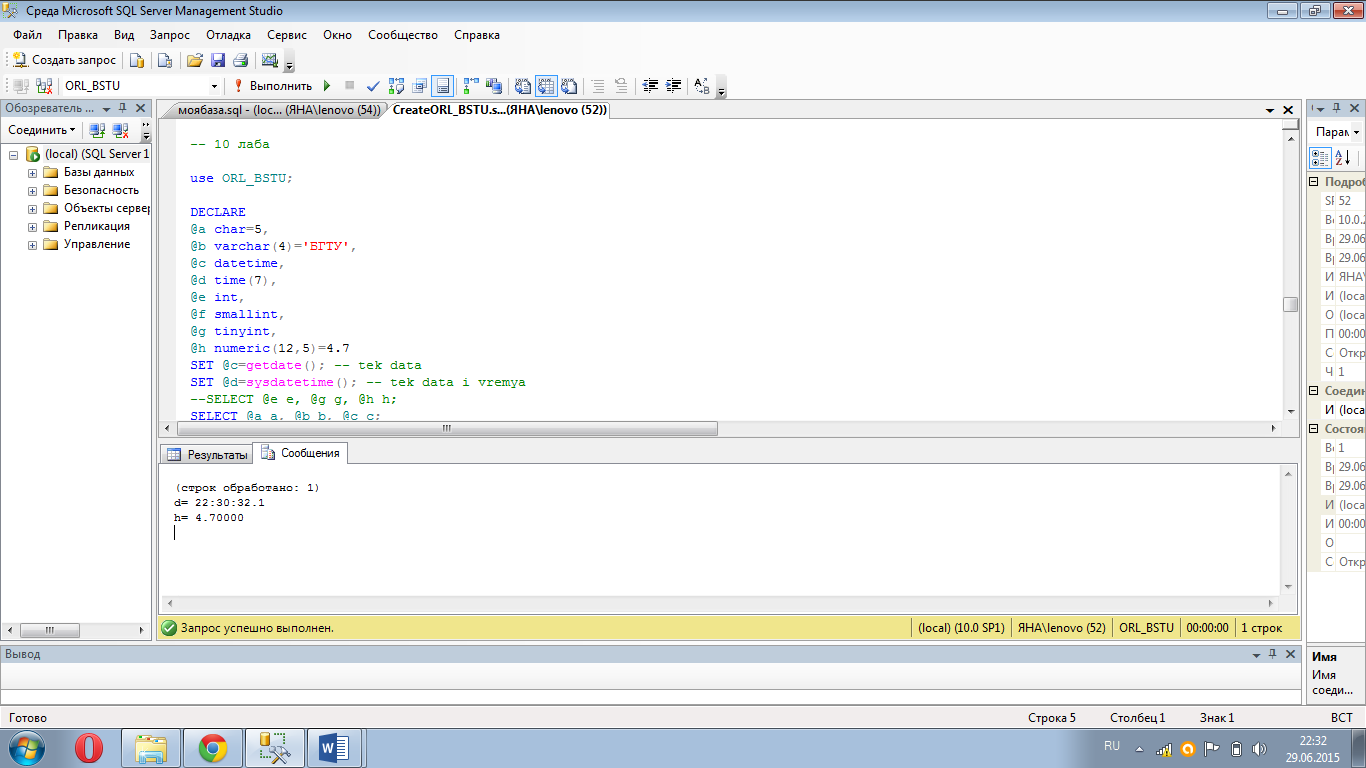
@b varchar(4)='БГТУ',

@c datetime,

@d time(7),

@e int,

@f smallint,

@g tinyint,

@h numeric(12,5)=4.7

SET @c=getdate(); -- tek data

SET @d=sysdatetime(); -- tek data i vremya

--SELECT @e e, @g g, @h h;

SELECT @a a, @b b, @c c;

print 'd= ' +cast(@d as varchar(10));

print 'h= ' +cast(@h as varchar(10));

DECLARE @a1 numeric(8,3)=(select CAST(sum(Вместимость)

as numeric(8,3))from AUDITORIUM),

@a2 real, @a3 numeric(8,3), @a4 real, @a5 numeric(8,3)

If @a1>200

begin

select @a2=(select CAST(count(\*) as numeric(8,3)) from AUDITORIUM),

@a3=(select CAST(AVG(Вместимость)as numeric(8,3)) from AUDITORIUM);

SET @a4=(select cast(count(\*) as numeric(8,3)) from AUDITORIUM where Вместимость<@a3)

select @a5=(select cast(count(\*) as numeric(8,3))/@a2\*100 from AUDITORIUM where Вместимость<@a3)

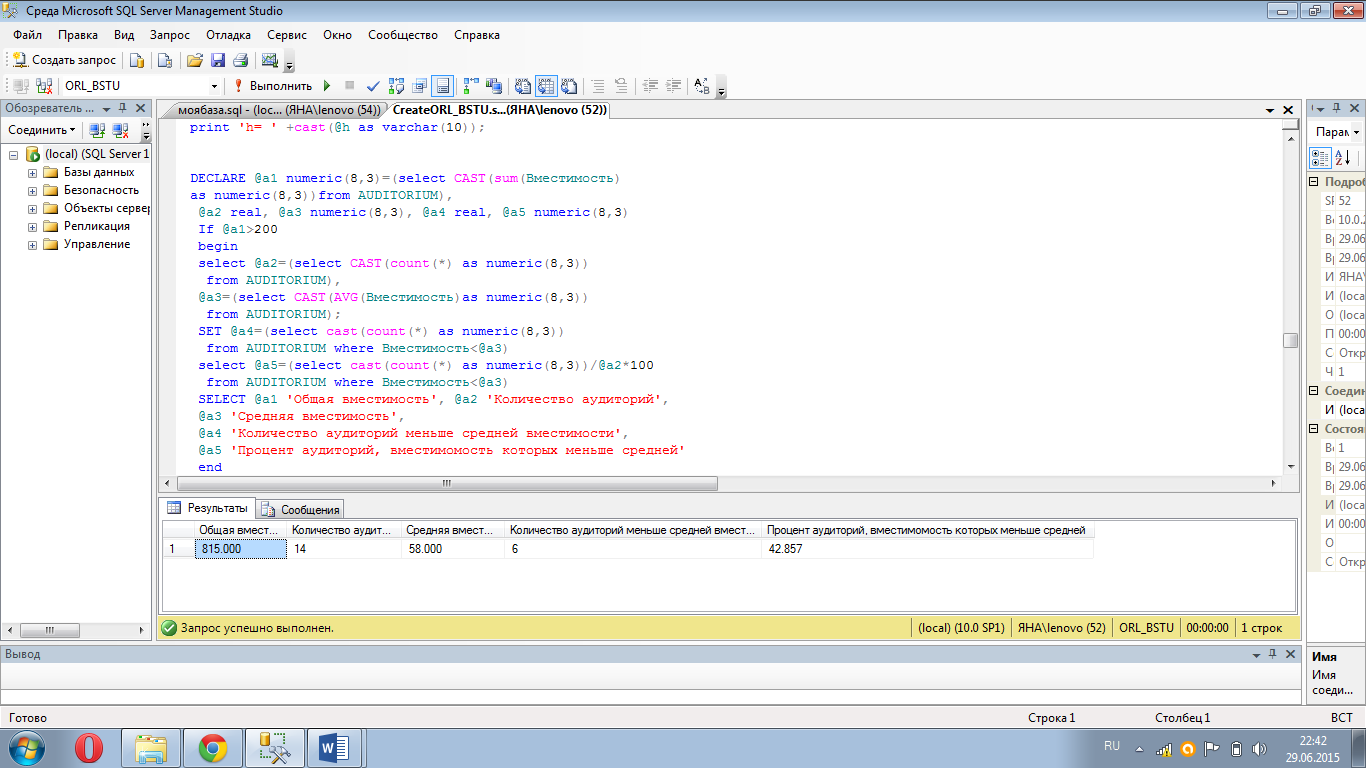
SELECT @a1 'Общая вместимость', @a2 'Количество аудиторий',

@a3 'Средняя вместимость',

@a4 'Количество аудиторий меньше средней вместимости',

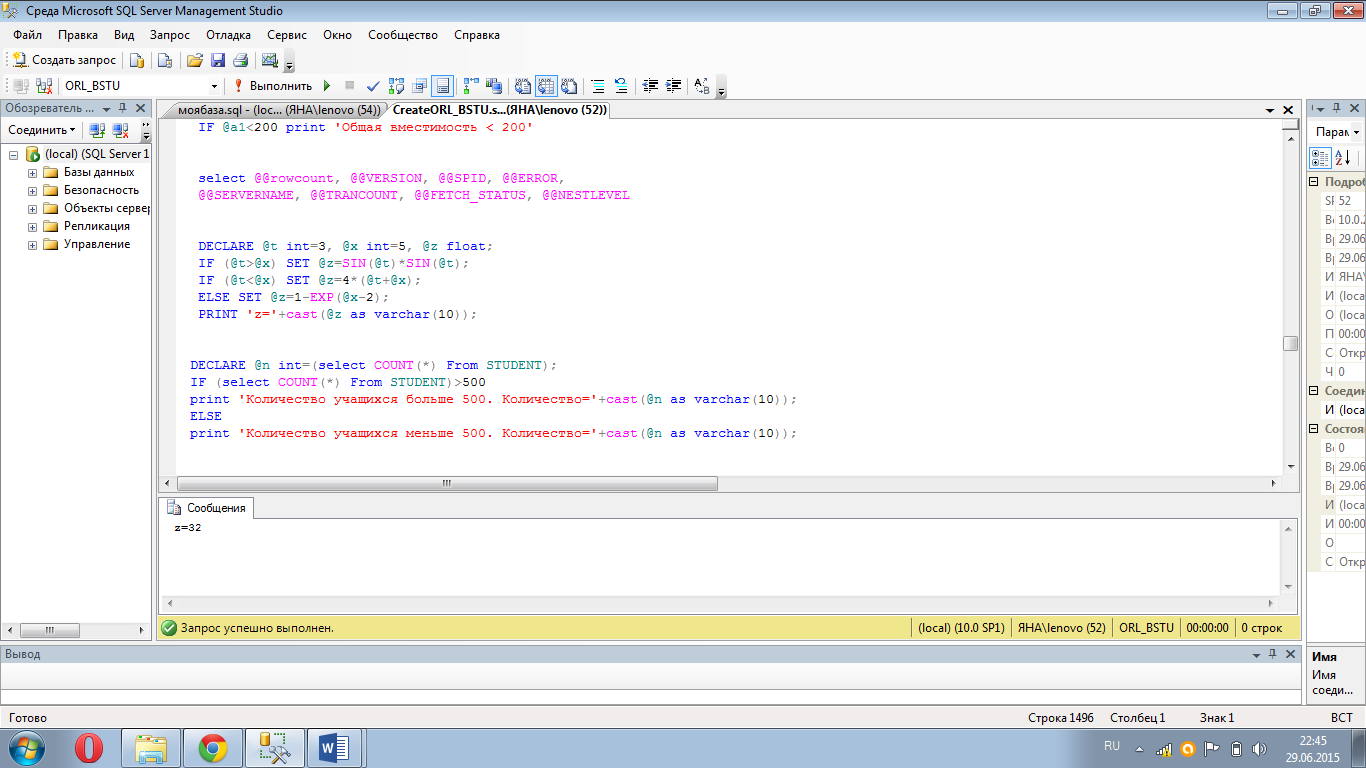
@a5 'Процент аудиторий, вместимомость которых меньше средней'

End IF @a1<200 print 'Общая вместимость < 200'



Вычисление функции:

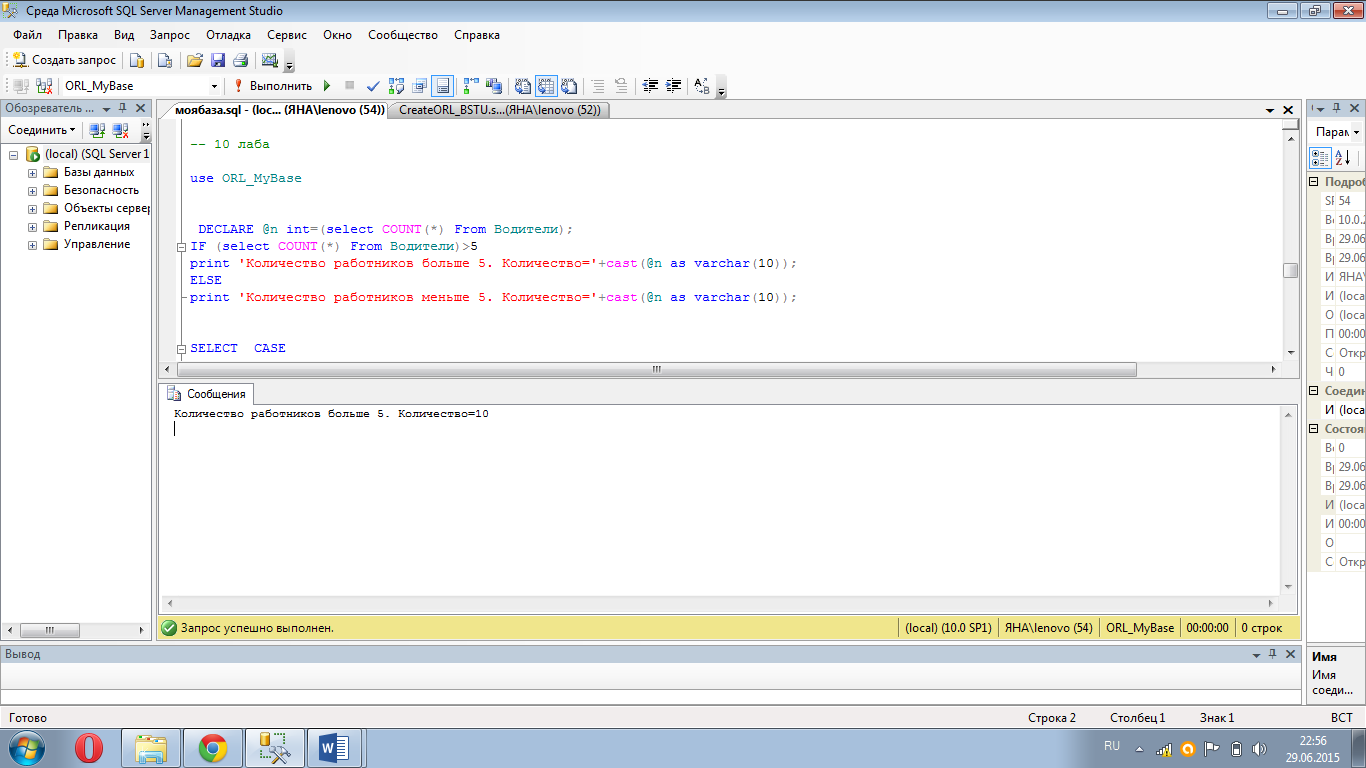
DECLARE @t int=3, @x int=5, @z float;

 IF (@t>@x) SET @z=SIN(@t)\*SIN(@t);

IF (@t<@x) SET @z=4\*(@t+@x);

ELSE SET @z=1-EXP(@x-2);

PRINT 'z='+cast(@z as varchar(10));

Для своей базы. Подсчет работников:

DECLARE @n int=(select COUNT(\*) From Водители);

IF (select COUNT(\*) From Водители)>5

print 'Количество работников больше 5. Количество='+cast(@n as varchar(10));

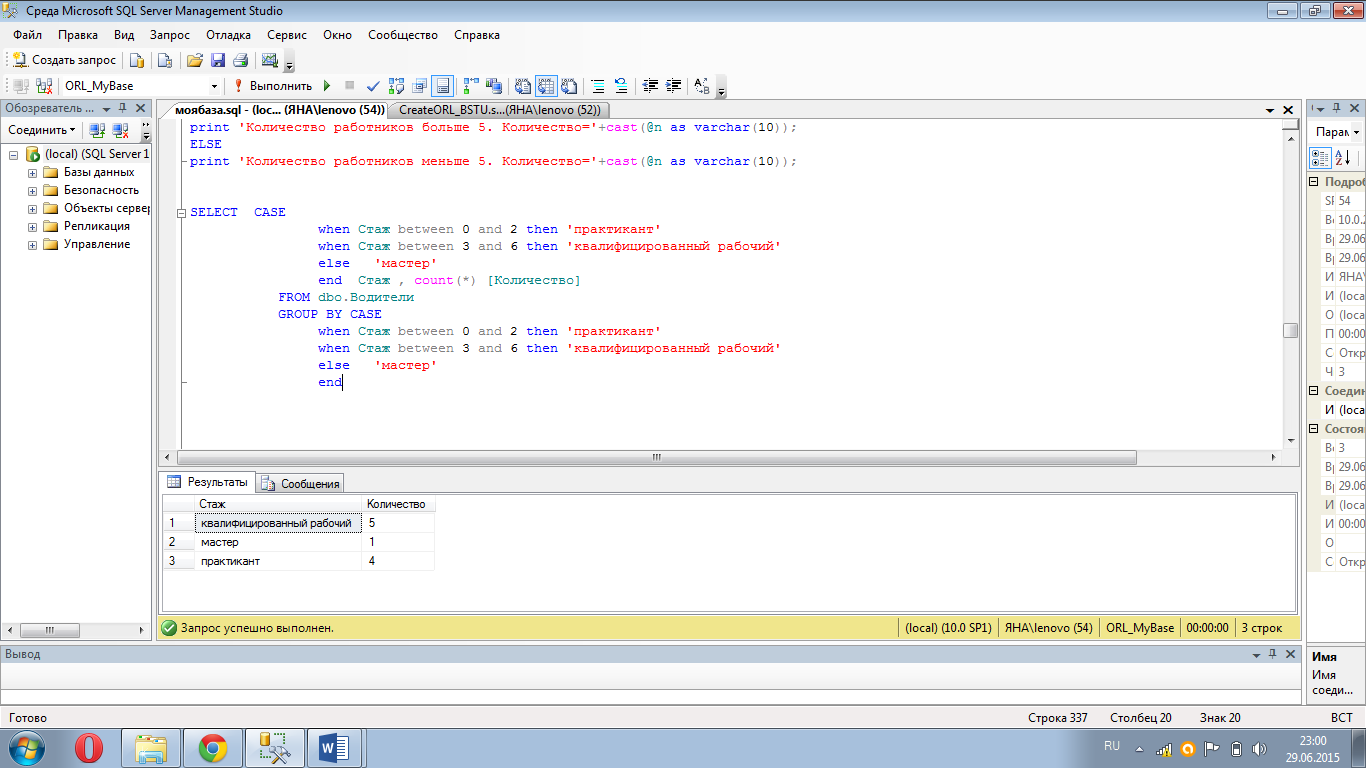
ELSE

print 'Количество работников меньше 5. Количество='+cast(@n as varchar(10));

И пределение их квалификации:

SELECT CASE

when Стаж between 0 and 2 then 'практикант'

 when Стаж between 3 and 6 then 'квалифицированный рабочий'

else 'мастер'

end Стаж , count(\*) [Количество]

FROM dbo.Водители

GROUP BY CASE

when Стаж between 0 and 2 then 'практикант'

when Стаж between 3 and 6 then 'квалифицированный рабочий'

else 'мастер'

end

**Лабораторная работа № 11. Создание и применение индексов**

Индекс – это объект базы данных, позволяющий ускорить поиск в определенной таблице. Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов. Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.

Создать временную локальную таблицу. Заполнить ее данными (не менее 1000 строк).

CREATE table #EXPLRE

( TIND int, TFIELD varchar(100) );

go

set nocount on; -- не выводить сообщения о вводе строк

declare @i int=0;

while @i<1000

begin

insert #EXPLRE(TIND, TFIELD)

values (FLOOR(5000\*RAND()),REPLICATE('stroka',10));

if (@i%1=0) print @i;

set @i=@i+1;

end;

go

select COUNT(\*) [kolichestvo strok] from #EXPLRE;

CREATE clustered index #EXPLRE\_CL on #EXPLRE(TIND asc) – кластеризованный

CREATE index #EX\_NONCLU on #EXPLR(TIND, CC) -- составной неуникальный, некластеризованный

CREATE index #EX\_TKEY on #EX(TKEY) INCLUDE (CC); -- некластеризованный покрытия

CREATE index #EXP\_WHERE on #EXP(TK) where (TK>=5000 and TK<15000); -- некластеризованный фильтруемый

Для своей:

С помощью системной процедуры SP\_HELPINDEX можно получить перечень индексов, связанных с заданной таблицей:

exec SP\_HELPINDEX'Водители';

SELECT

Водители.Ф\_И\_О, Водители.Стаж, Маршруты.Оплата, Поездки.Дата\_отправления

FROM (Водители Inner join Поездки ON Водители.Код\_водителя\_ФИО=Поездки.Код\_водителя\_ФИО)

Inner join Маршруты ON Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута

ORDER BY

(Case when (Водители.Стаж=7) then 1

when (Водители.Стаж=5) then 2

else 3

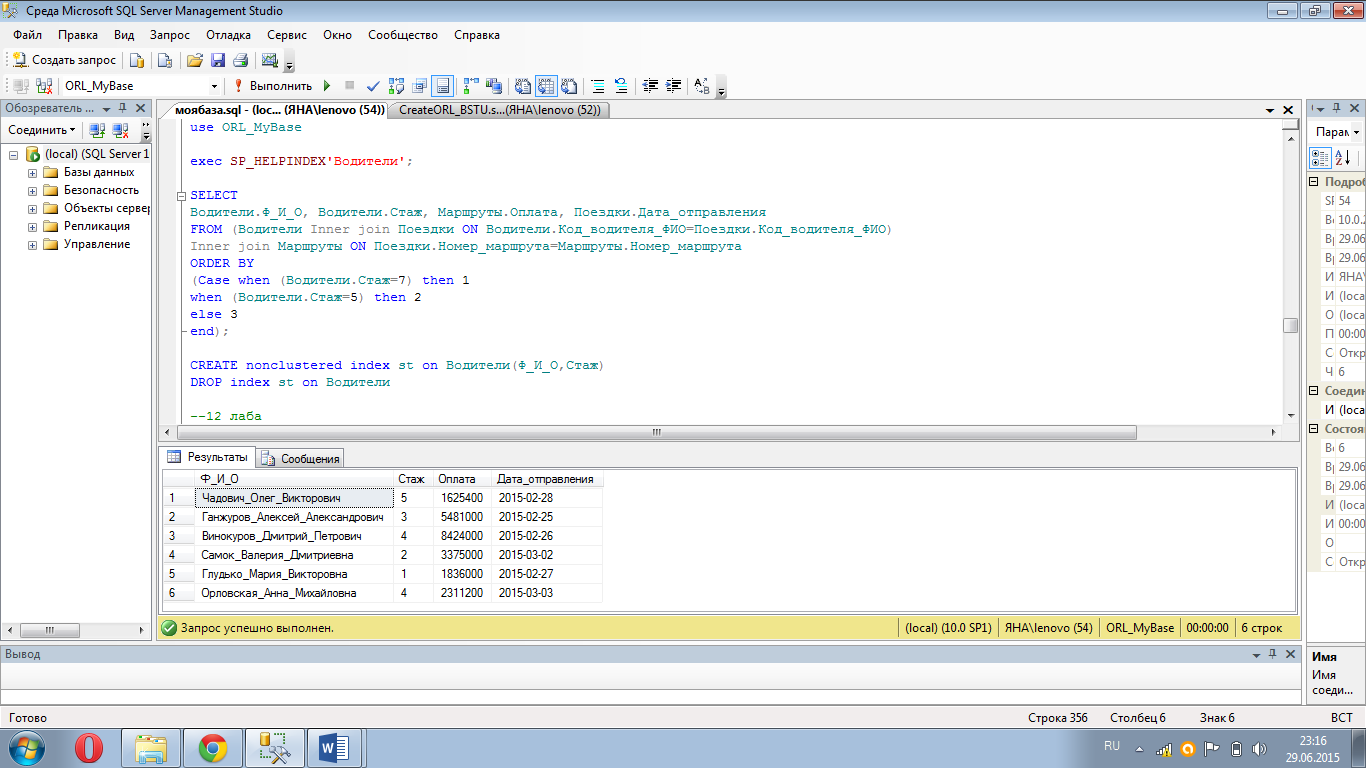
end);

Создадим некластеризованный индекс:

CREATE nonclustered index st on Водители(Ф\_И\_О,Стаж)

Удалим индекс:

DROP index st on Водители



**Лабораторная работа № 12. Обработка результатов запросов с помощью курсоров**

Курсор является программной конструкцией, которая служит для хранения результата запроса и дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают локальные и гло-бальные (по умолчанию), статические и динамические (по умолчанию). Работа с курсором осуществляется в следующей последовательности: курсор объявляется в операторе DECLARE, открывается с помощью оператора OPEN, с помощью оператора FETCH считывается одна или несколько строк результирующего набора. Результат каждого считывания про-веряется с помощью системной функции @@FETCH\_STATUS. Затем курсор закрывается с помощью оператора CLOSE. Если курсор глобальный, то он должен быть освобожден с помощью оператора DEALLOCATE.

Сценарий, формирующий список дис-циплин на кафедре ИСиТ:

DECLARE @kaf char(20), @k char(300)=' ';

DECLARE dis CURSOR local for select Код\_дисциплины from SUBJECT

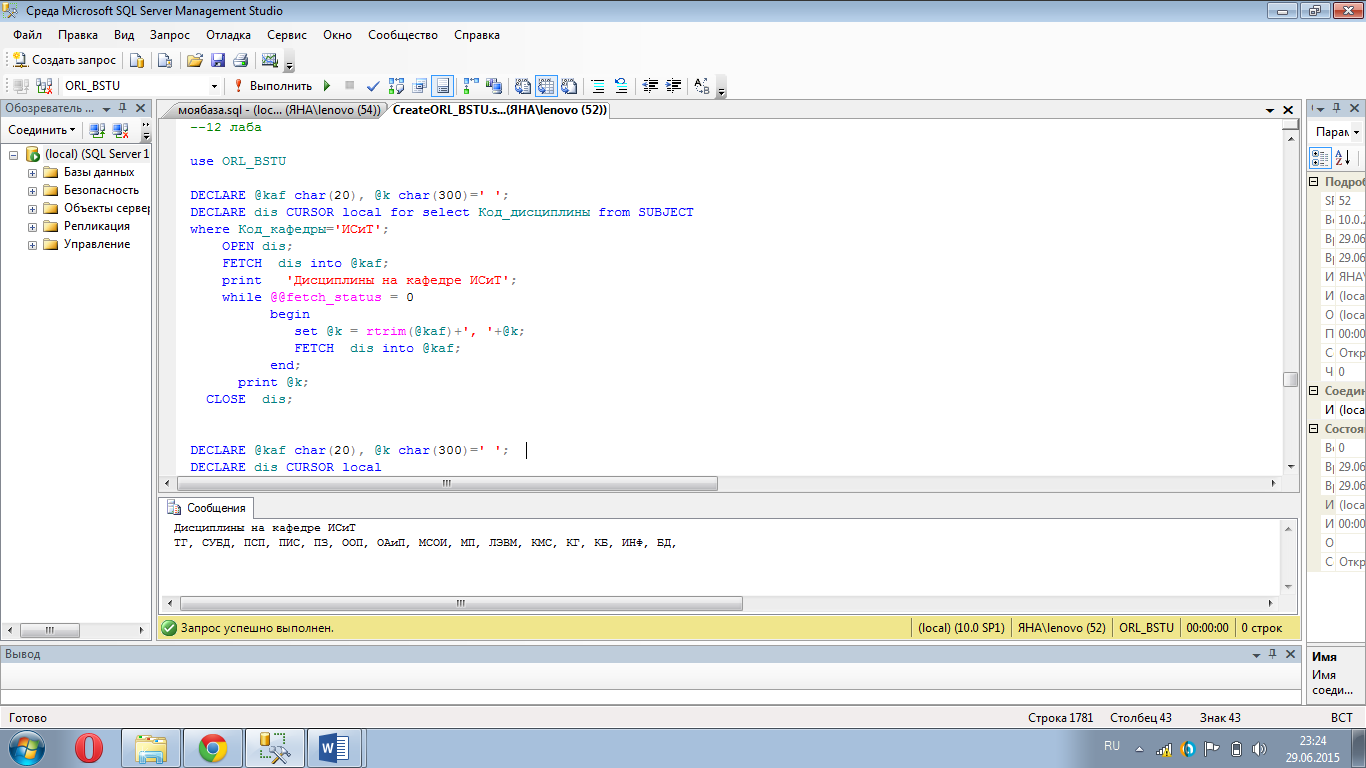
where Код\_кафедры='ИСиТ';

OPEN dis;

FETCH dis into @kaf;

print 'Дисциплины на кафедре ИСиТ';

while @@fetch\_status = 0

 begin

set @k = rtrim(@kaf)+', '+@k;

FETCH dis into @kaf;

end;

print @k;

CLOSE dis;

DECLARE di CURSOR global

declare c\_teacher cursor local static

declare c\_teacher cursor local dynamic

DECLARE @kaf char(20), @k char(300)=' ';

DECLARE dis CURSOR local for select Дата\_отправления from Поездки

OPEN dis;

FETCH dis into @kaf;

print 'Рейсы';

while @@fetch\_status = 0

begin

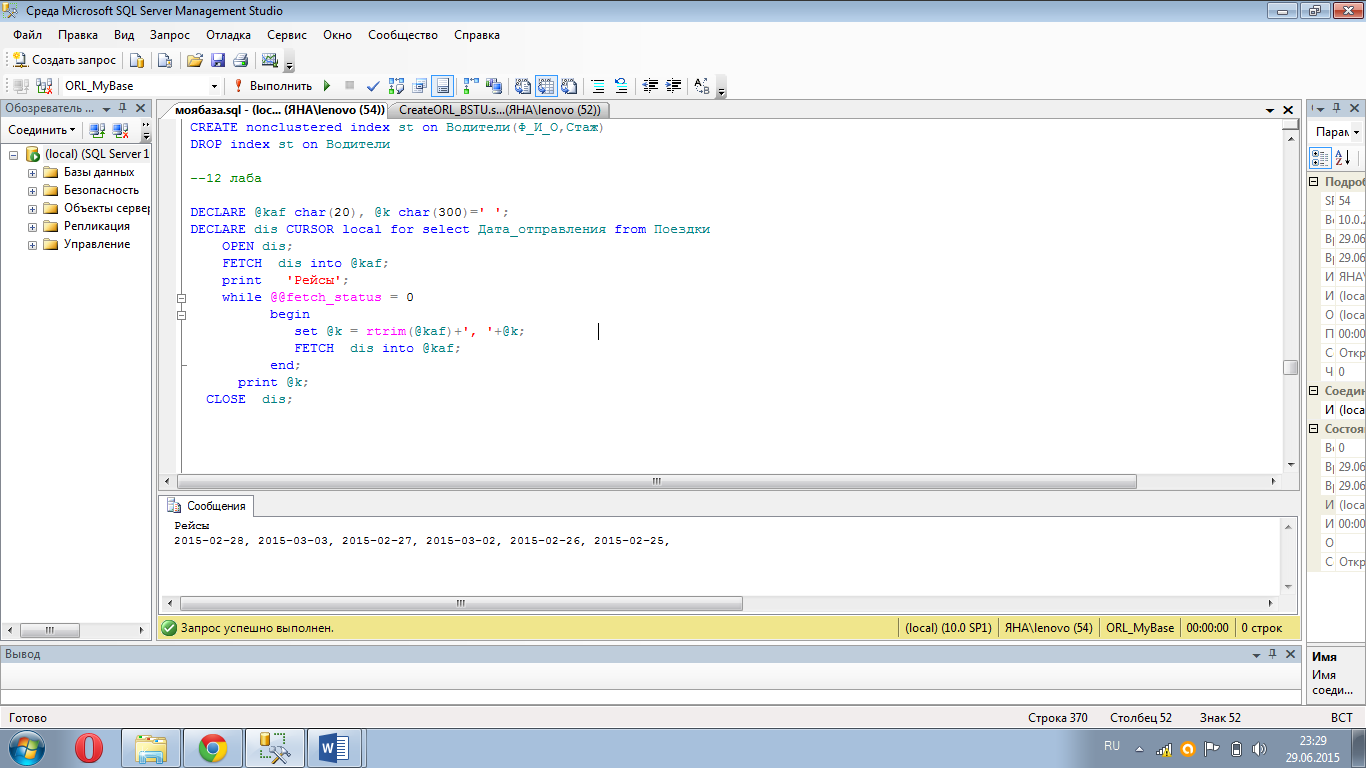
set @k = rtrim(@kaf)+', '+@k;

FETCH dis into @kaf;

end;

print @k;

CLOSE dis;



**Лабораторная работа № 13. Особенности использования транзакций**

Транзакция - это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменя-ющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполни-лись.

Основные свойства транзакции: атомарность (операторы изменения БД, включенные в транзакцию, либо выполнят-ся все, либо не выполнится ни один); согласованность (транзакция должна фиксировать новое согласованное состояние БД); изолированность (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения); долго-вечность (изменения в БД, выполненные и зафиксированные транзакцией, могут быть отменены только с помощью новой транзакции).

Сценарий, демонстрирующий работу в режиме неявной транзакции:

declare @c int, @flag char = 'c'; -- оператор фиксации commit или отката rollback?

SET IMPLICIT\_TRANSACTIONS ON -- включ. режим неявной транзакции

-- create table X(K int ); -- начало транзакции

insert X values (1),(2),(3);

set @c = (select count(\*) from X);

print 'количество строк в таблице X: ' + cast( @c as varchar(2));

if @flag = 'c' commit; -- завершение транзакции: фиксация

else rollback; -- завершение транзакции: откат

SET IMPLICIT\_TRANSACTIONS OFF -- выключ. режим неявной транзакции

-- действует режим автофиксации

if exists (select \* from SYS.OBJECTS -- таблица X есть?

where OBJECT\_ID= object\_id(N'DBO.X') )

print 'таблица X есть';

else print 'таблицы X нет'

Сценарий, демонстрирующий работу в режиме явной транзакции:

go

begin try

begin transaction -- начало явной транзакции

delete SUBJECT where Код\_дисциплины='ДМ';

insert SUBJECT values ('ТГ', 'Теория графов','ИСиТ' ),

('КБ', 'Комбинаторика','ИСиТ' );

commit transaction; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_SUBJECT%', error\_message()) > 0

then 'нет кафедры с таким кодом'

when error\_number() = 2627 and patindex('%PK\_SUBJECT%', error\_message()) > 0

then 'дублирование кода информации'

else 'неизвестная ошибка: '+ cast(error\_number() as varchar(5))+ error\_message()

end;

if @@trancount > 0 rollback transaction ; -- откат транзакции

end catch;

declare @point varchar(32); -- макс. длина имени 32

begin try

begin tran -- начало явной транзакции

delete SUBJECT where Код\_дисциплины='ДМ';

set @point = 'p1'; save tran @point; -- контрольная точка p1

insert SUBJECT values ('ТГ', 'Теория графов','ИСиТ' );

set @point = 'p2'; save tran @point; -- контрольная точка p2

insert SUBJECT values ('КБ', 'Комбинаторика','ИСиТ' );

commit tran; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_SUBJECT%', error\_message()) > 0

then 'нет кафедры с таким кодом'

when error\_number() = 2627 and patindex('%PK\_SUBJECT%', error\_message()) > 0

then 'дублирование информации'

else 'неизвестная ошибка: '+ cast(error\_number() as varchar(5)) + error\_message()

end;

if @@trancount > 0

begin

print 'контрольная точка: '+ @point;

rollback tran @point; -- откат к контрольной точке

commit tran; -- фиксация изменений, выполненных до контрольной точки

end;

end catch;

-- A ---

set transaction isolation level READ UNCOMMITTED -- неподтвержденное чтение

begin transaction

-------------------------- t1 ------------------

select @@SPID, 'insert FACULTY' 'результат', \* from FACULTY

where Код\_факультета='ИТ';

select @@SPID, 'update FACULTY' 'результат', \* from PULPIT

where Код\_факультета='ИТ';

commit;

-------------------------- t2 -----------------

--- B --

begin transaction

select @@SPID

insert FACULTY values ('ИТ', 'Информационных Технологий');

update PULPIT set Код\_факультета = 'ИТ'

where Код\_кафедры= 'ИСиТ'

-------------------------- t1 --------------------

-------------------------- t2 --------------------

rollback;

-- A ---

set transaction isolation level READ COMMITTED -- неповторяющееся чтение

begin transaction

select count(\*) from PULPIT

where Код\_факультета = 'ИТ';

-------------------------- t1 ------------------

-------------------------- t2 -----------------

select 'update PULPIT' 'результат', count(\*)

from PULPIT where Код\_факультета = 'ИТ';

commit;

--- B ---

begin transaction

-------------------------- t1 --------------------

update PULPIT set Код\_факультета = 'ИТ'

where Код\_кафедры = 'ИСиТ'

commit;

-- A ---

set transaction isolation level REPEATABLE READ -- фантомное чтение

begin transaction

select Код\_преподавателя from TEACHER where Код\_кафедры = 'ПОиСОИ';

-------------------------- t1 ------------------

-------------------------- t2 -----------------

select case

when Код\_преподавателя = 'ПТР' then 'insert TEACHER' else ' '

end 'результат', Код\_преподавателя from TEACHER where Код\_кафедры = 'ПОиСОИ';

commit;

--- B ---

begin transaction

-------------------------- t1 --------------------

insert TEACHER values ('ПТР', 'Петров Петр Петрович', 'м', 'ПОиСОИ');

commit;

-------------------------- t2 --------------------

Для своей базы:

go

begin try

begin transaction -- начало явной транзакции

delete Поездки where Код\_водителя\_ФИО='ОМВ';

delete Водители where Код\_водителя\_ФИО='ОМВ';

insert Водители values ('ОЯВ', 'Орловская Яна Владимировна',1 ),

('ВАВ', 'Винокурова Анастасия Владимировна',1 );

commit transaction; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_Поездки%', error\_message()) > 0

then 'нет такого водителя'

when error\_number() = 2627 and patindex('%PK\_Водители%', error\_message()) > 0

then 'дублирование кода информации'

else 'неизвестная ошибка: '+ cast(error\_number() as varchar(5))+ error\_message()

end;

if @@trancount > 0 rollback transaction ; -- откат транзакции

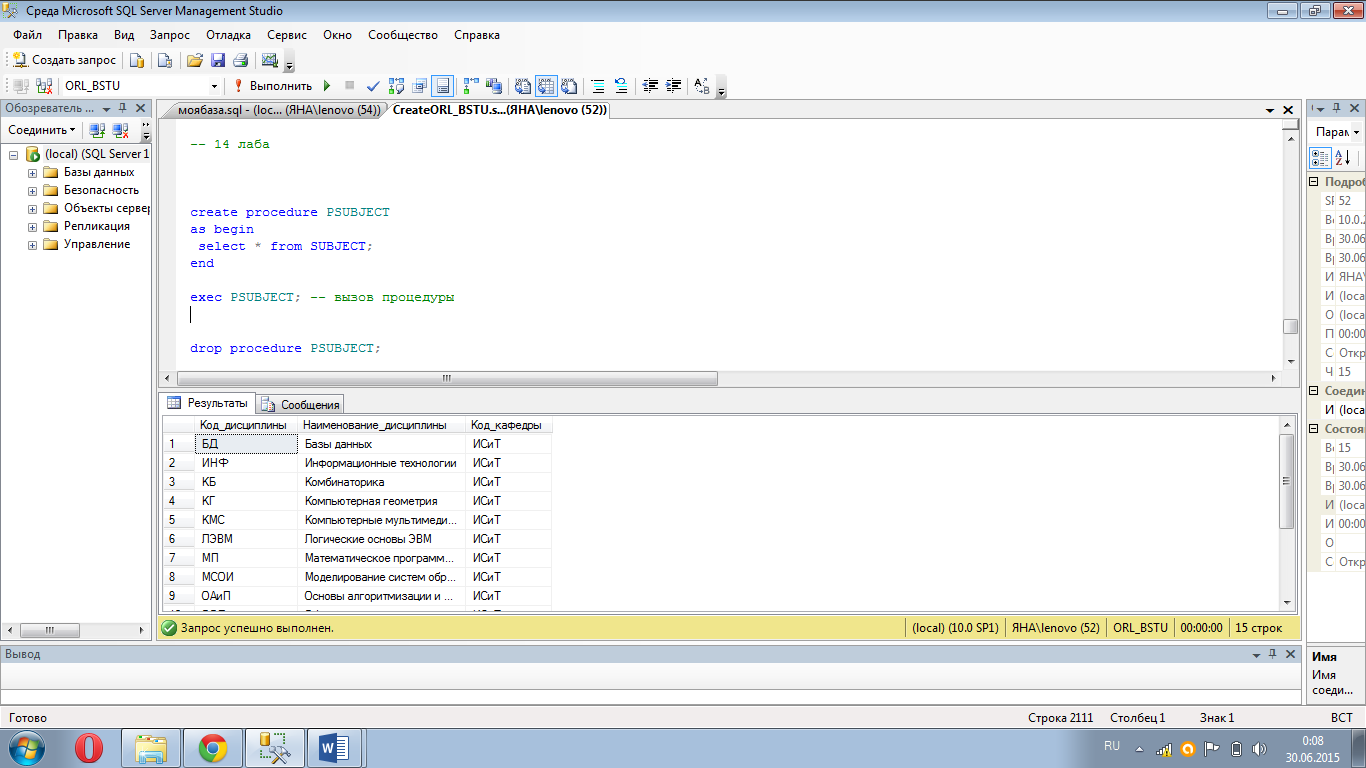
end catch;

**Лабораторная работа № 14. Разработка хранимых процедур**

Хранимая процедура – это объект базы данных, представляющий собой поименованный код T-SQL. Хранимая про-цедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. Процедура может принимать входные и формировать выходные параметры. Результатом ее выполнения может быть це-лочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирую-щих наборов, сформированных операторами SELECT, либо содержимое стандартного выходного потока, полученного при выполнении операторов PRINT. Вызов процедуры осуществляется оператором EXECUTE (EXEC).

В хранимых процедурах допускается применение основных DDL и всех DML и TCL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

Создание процедуры:

create procedure PSUBJECT

as begin

select \* from SUBJECT;

end

exec PSUBJECT; -- вызов процедуры

Удаление процедуры:

drop procedure PSUBJECT;

alter procedure PSUBJECT -- модификация процедуры

@p varchar(20),

@c int output

as begin

declare @rc int = (select COUNT(\*) from SUBJECT);

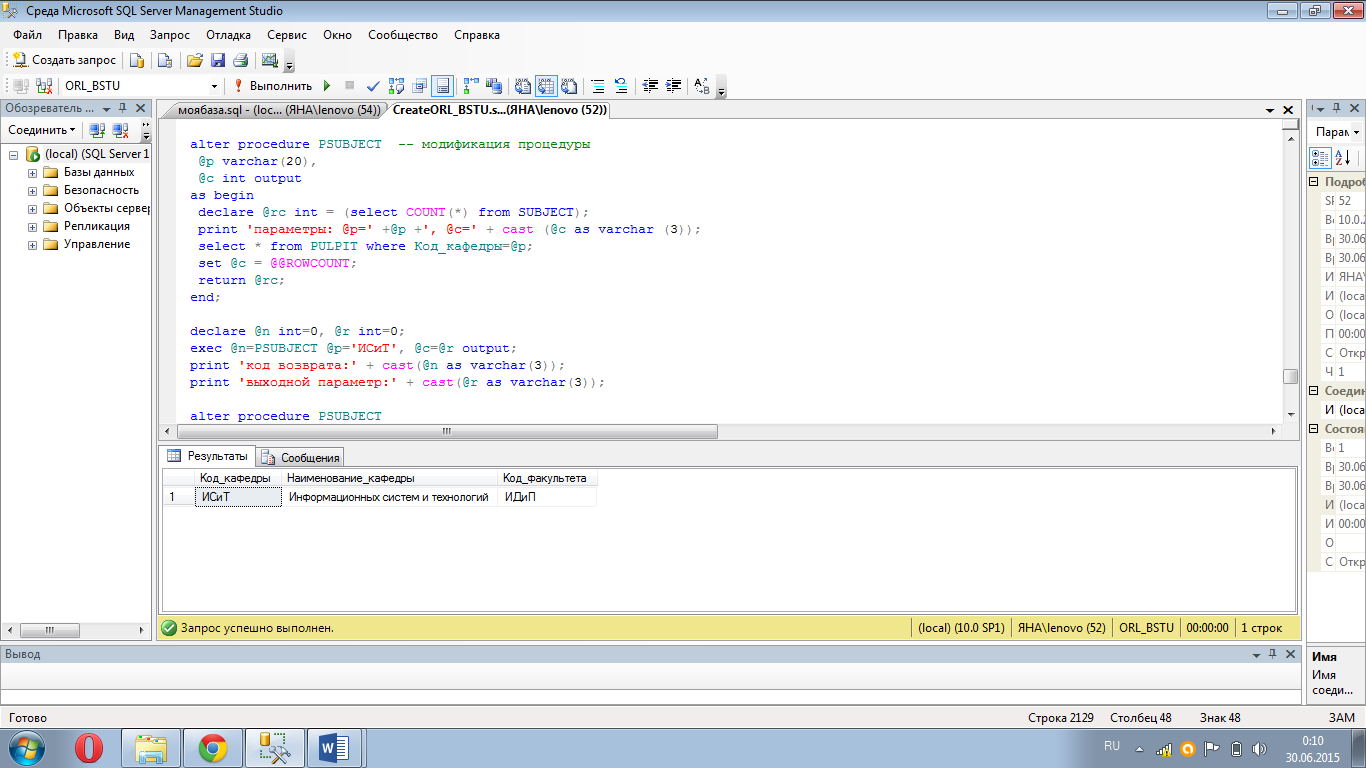
print 'параметры: @p=' +@p +', @c=' + cast (@c as varchar (3));

select \* from PULPIT where Код\_кафедры=@p;

set @c = @@ROWCOUNT;

return @rc;

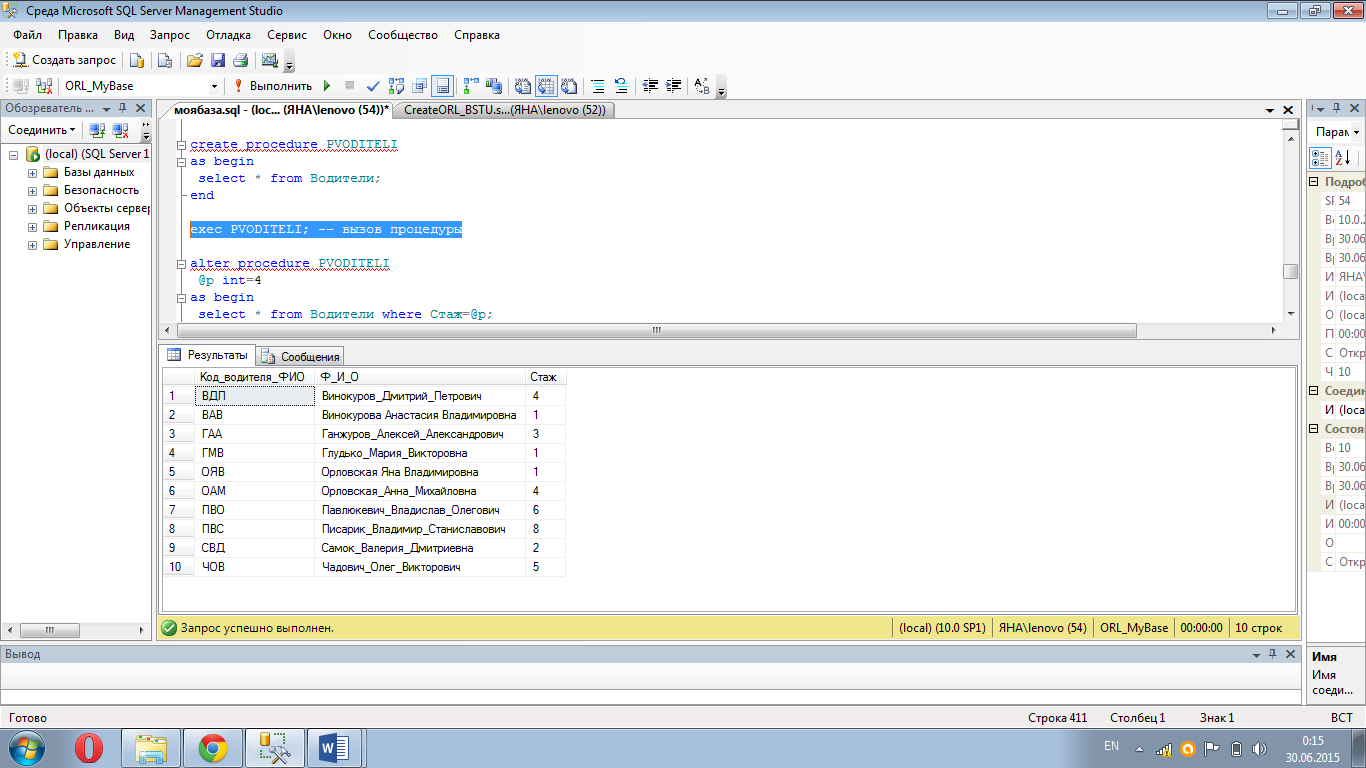
end;

declare @n int=0, @r int=0;

exec @n=PSUBJECT @p='ИСиТ', @c=@r output;

print 'код возврата:' + cast(@n as varchar(3));

print 'выходной параметр:' + cast(@r as varchar(3));

Для своей базы:

create procedure PVODITELI

as begin

select \* from Водители;

end

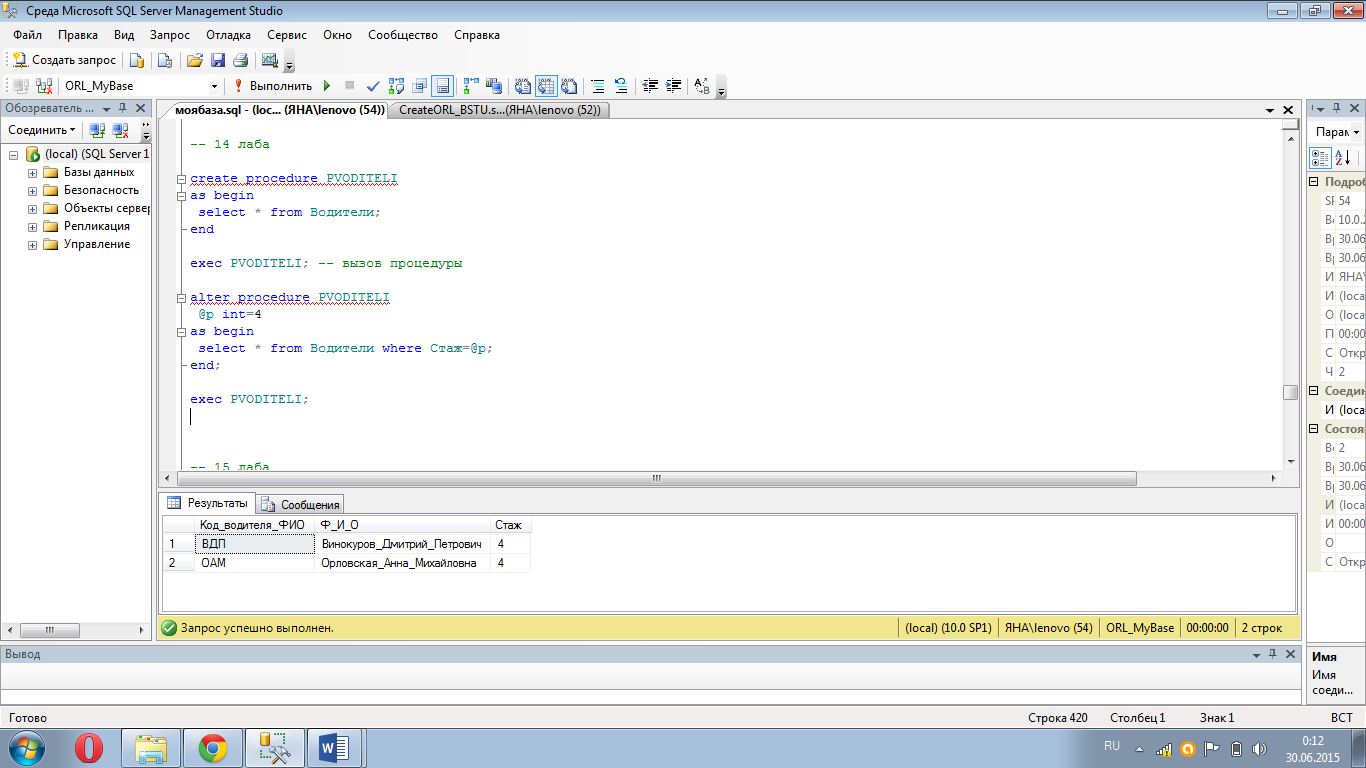
exec PVODITELI; -- вызов процедуры

alter procedure PVODITELI

@p int=4

as begin

select \* from Водители where Стаж=@p;

end;

**Лабораторная работа № 15. Разработка и использование функций**

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций надо использовать операторы CREATE, DROP и ALTER соответственно. Отличие функций от хранимых про-цедур в ограничениях, накладываемых на код функции, в форме представления результата работы, а также в способе вызова. В функции не допускается применение DDL-операторов, DML-операторов, изменяющих БД (INSERT, DELETE, UPDATE), конструкций TRY/CATCH, а также использование транзакций.

Результатом выполнения функции является возвращаемое к точке вызова значение. Если функция возвращает един-ственное значение (число, строка, дата, время и пр.), то она называется скалярной. Функция, возвращающая таблицу, называется табличной. В зависимости от структуры кода, различают встроенные функции и многооператорные таблич-ные функции.

create function COUNT\_STUDENTS(@faculty varchar(20)) returns int -- скалярная функция

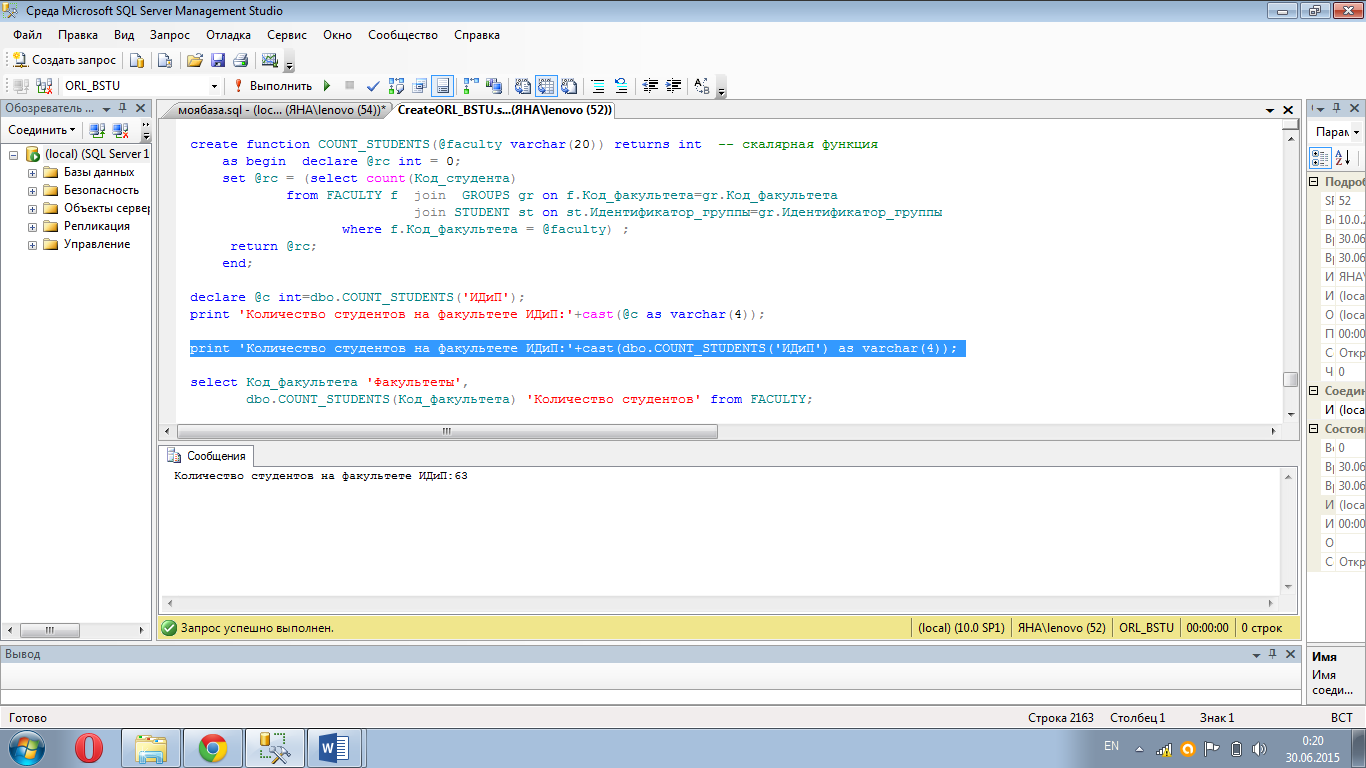
as begin declare @rc int = 0;

set @rc = (select count(Код\_студента)

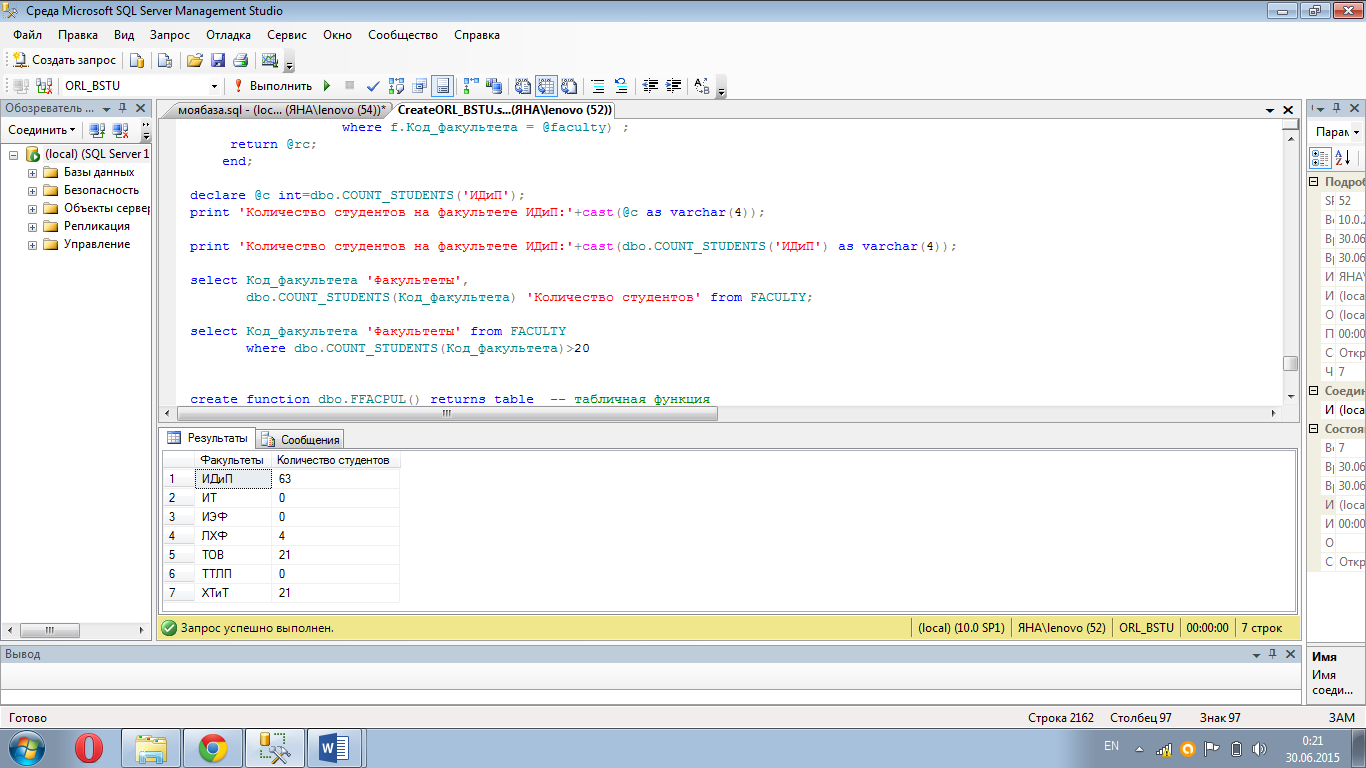
from FACULTY f join GROUPS gr on f.Код\_факультета=gr.Код\_факультета

join STUDENT st on st.Идентификатор\_группы=gr.Идентификатор\_группы

where f.Код\_факультета = @faculty) ;

 return @rc;

end;

declare @c int=dbo.COUNT\_STUDENTS('ИДиП');

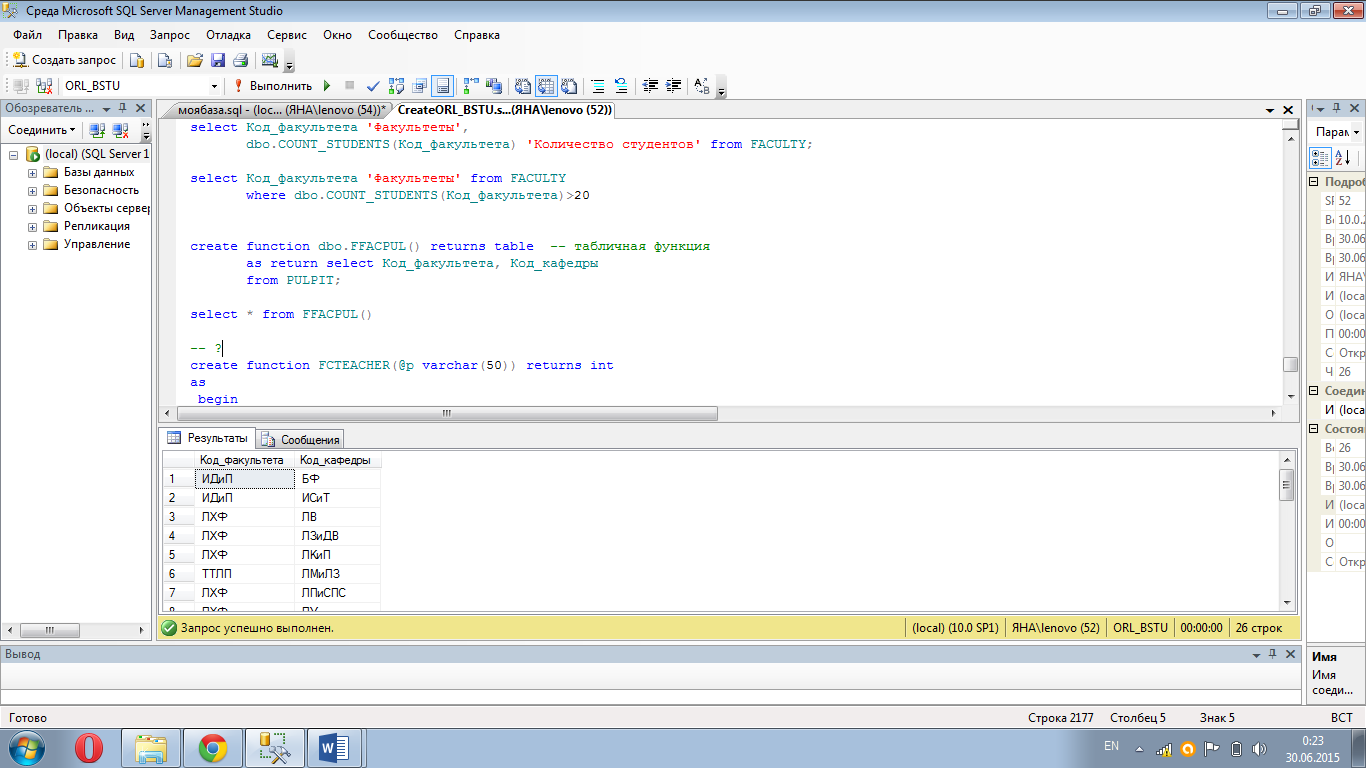
print 'Количество студентов на факультете ИДиП:'+cast(@c as varchar(4));

select Код\_факультета 'Факультеты',

dbo.COUNT\_STUDENTS(Код\_факультета) 'Количество студентов' from FACULTY;

select Код\_факультета 'Факультеты' from FACULTY

where dbo.COUNT\_STUDENTS(Код\_факультета)>20

create function dbo.FFACPUL() returns table -- табличная функция

as return select Код\_факультета, Код\_кафедры

from PULPIT;

select \* from FFACPUL()

Для своей базы:

create function COUNT\_VODITELI(@vod varchar(20)) returns int -- скалярная функция

as begin declare @rc int = 0;

set @rc = (select count(Код\_водителя\_ФИО) from Маршруты m join Поездки p on m.Номер\_маршрута=p.Номер\_маршрута

where m.Номер\_маршрута = @vod) ;

return @rc;

end;

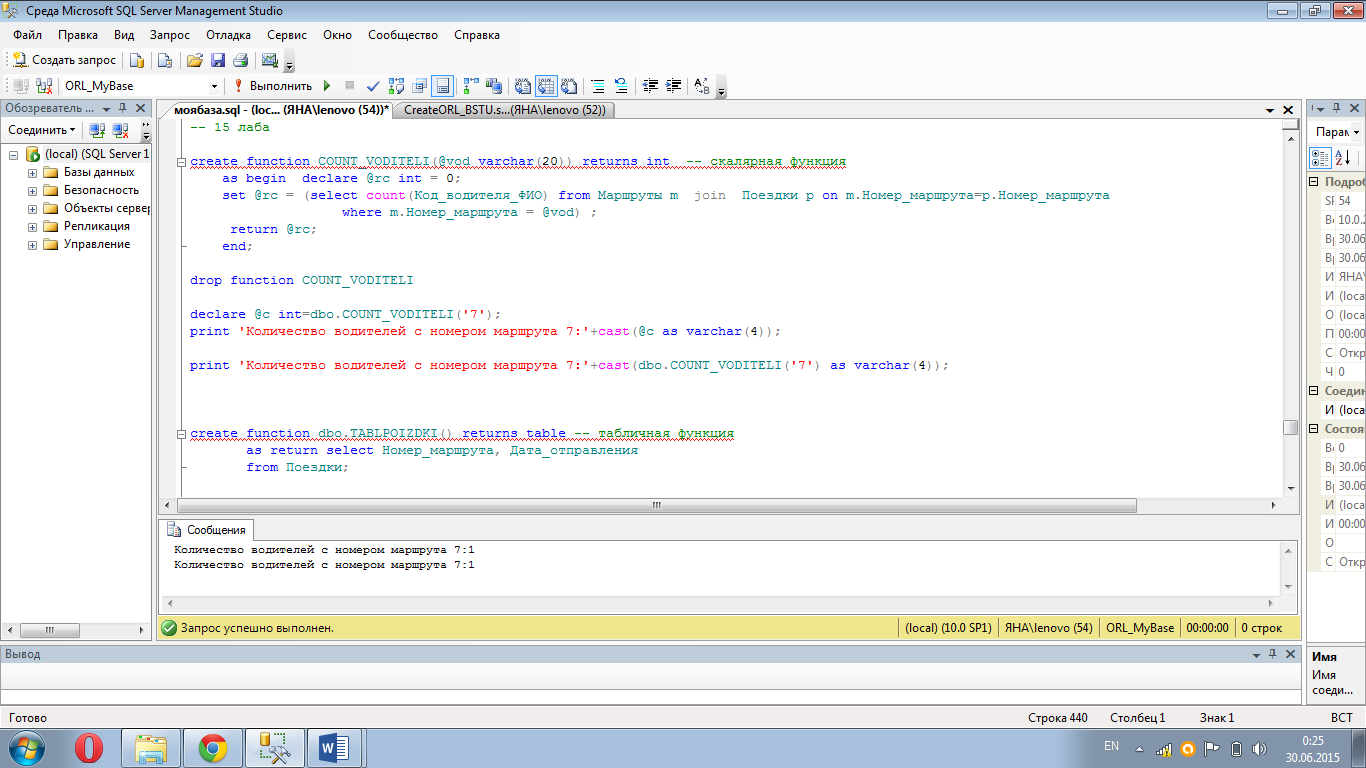
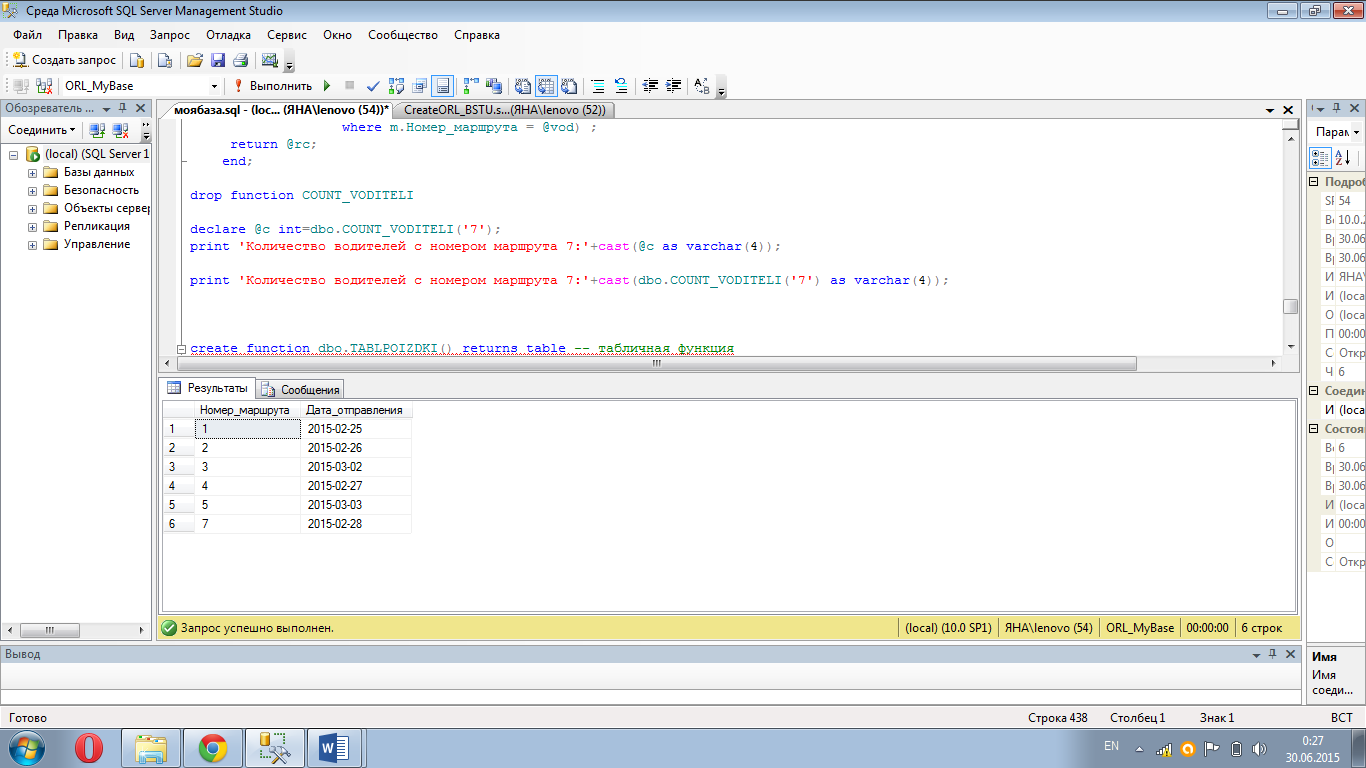
drop function COUNT\_VODITELI

declare @c int=dbo.COUNT\_VODITELI('7');

print 'Количество водителей с номером маршрута 7:'+cast(@c as varchar(4));

или

print 'Количество водителей с номером маршрута 7:'+cast(dbo.COUNT\_VODITELI('7') as varchar(4));



create function dbo.TABLPOIZDKI() returns table -- табличная функция

as return select Номер\_маршрута, Дата\_отправления

from Поездки;

select \* from TABLPOIZDKI()

**Лабораторная работа № 16. Применение DML-триггеров**

Триггер – это особый вид хранимой процедуры, предназначенной для обработки событий в БД. Поддерживается два типа триггеров: DDL-триггеры и DML-триггеры. Для каждого типа определено свое семейство событий, обработку ко-торых триггер этого типа может выполнять.

create table TR\_MARSHRUTS

( ID int identity,

STMT varchar(20) check (STMT in ('INS','DEL','UPD')),

TRNAME varchar(50),

CC varchar(300) )

Триггер реагирующий на вставку:

create trigger TR\_MARSHRUTS\_INS

on Маршруты after insert

as declare @a1 int, @a2 int, @a3 int, @a4 real, @in varchar(300);

print 'Insert operation';

set @a1=(select[Номер\_маршрута] from INSERTED);

set @a2=(select[Дальность] from INSERTED);

set @a3=(select[Количество\_дней\_в\_пути] from INSERTED);

set @a4=(select[Оплата] from INSERTED);

set @in=@a1+''+@a2+''+@a3+''+@a4;

insert into TR\_MARSHRUTS(STMT,TRNAME,CC)

values('INS','TR\_MARSHRUTS\_INS',@in);

return;

go

insert into Маршруты(Номер\_маршрута,Дальность,Количество\_дней\_в\_пути,Оплата)

values(11,230,4,3250600);

select \* from TR\_MARSHRUTS

select \* from Маршруты

drop table TR\_MARSHRUTS

**Лабораторная работа № 17. Использование XML**

XML (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена дан-ными между компонентами информационных систем. При работе с базами данных важными являются две задачи: пре-образование табличных данных в XML-структуры и преобразование XML-структур в строки реляционной таблицы.

select \* from Маршруты where Количество\_дней\_в\_пути='5'

select \* from Маршруты where Количество\_дней\_в\_пути='4' for xml raw

select \* from Маршруты where Количество\_дней\_в\_пути='3' for xml auto

select v.Ф\_И\_О, p.Дата\_отправления

from Водители v join Поездки p

on v.Код\_водителя\_ФИО=p.Код\_водителя\_ФИО

where p.Дата\_отправления in ('2015-02-26','2015-02-25') order by p.Дата\_отправления

for xml auto

